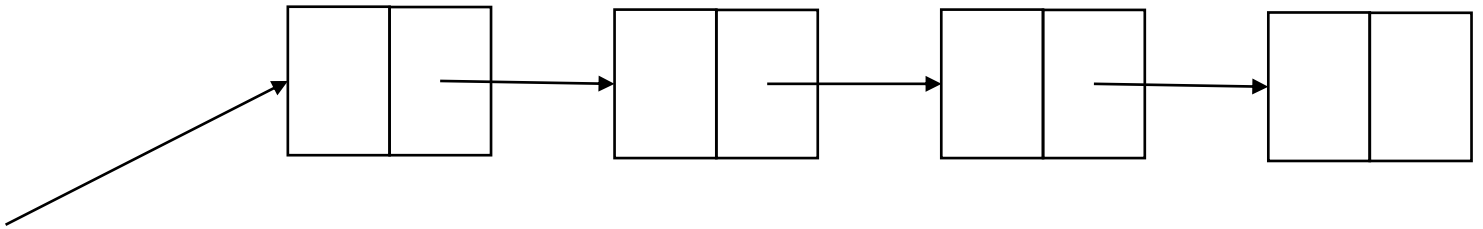
Introduction to Tree Data Structures

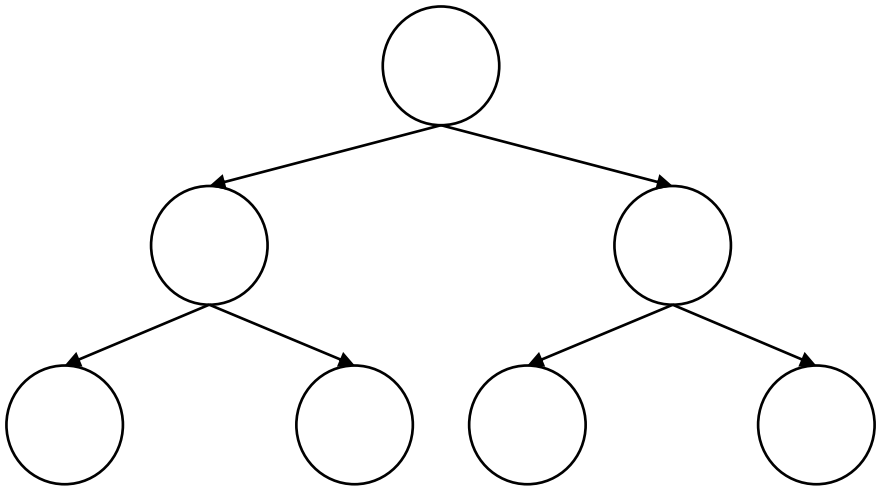
Linear Data Structures

- Till now, the data structures we have studied all have a common property.
- They all store data in a **linear** sequence, i.e., a data element may appear before other elements, and after some other elements.
- Example: Singly-linked lists.



Trees: A Hierarchical Data Structure

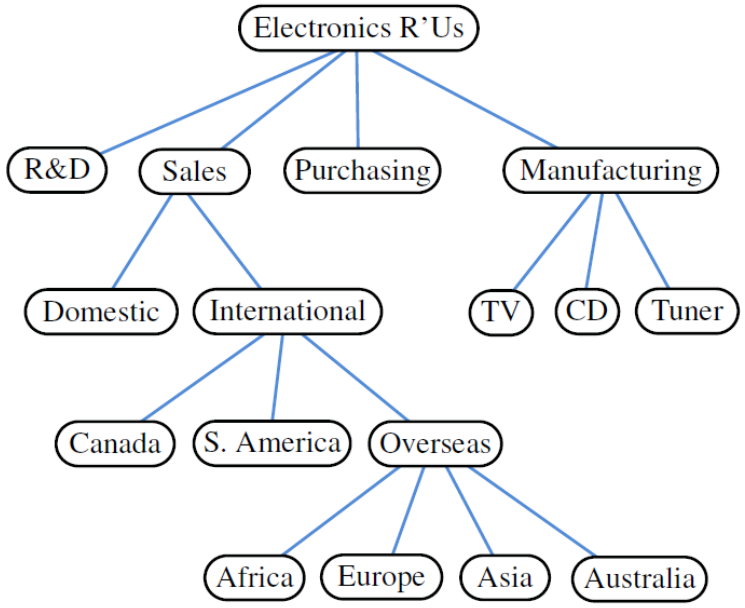
- A tree is a data structure that stores information **hierarchically**.



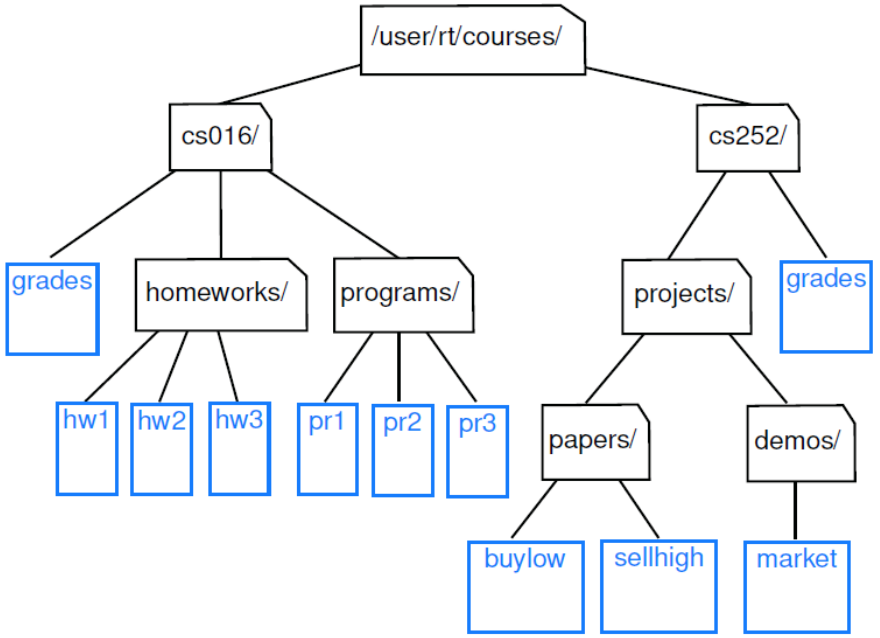
- Thus, in trees some data elements are located “above” other elements and “below” certain other elements.

Benefits?

1. Representing data that is naturally hierarchical.



A tree representing the organizational structure of a company.

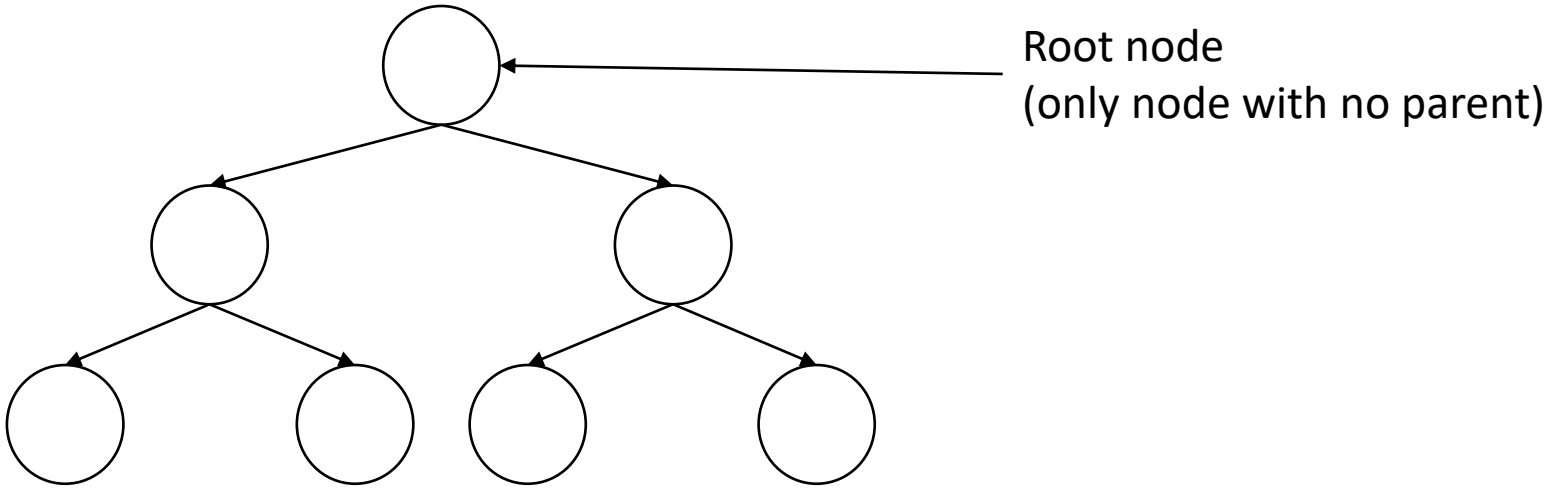


A tree representing a portion of a file system.

2. Efficient algorithm implementation (as compared to linear structures).

Tree Definition

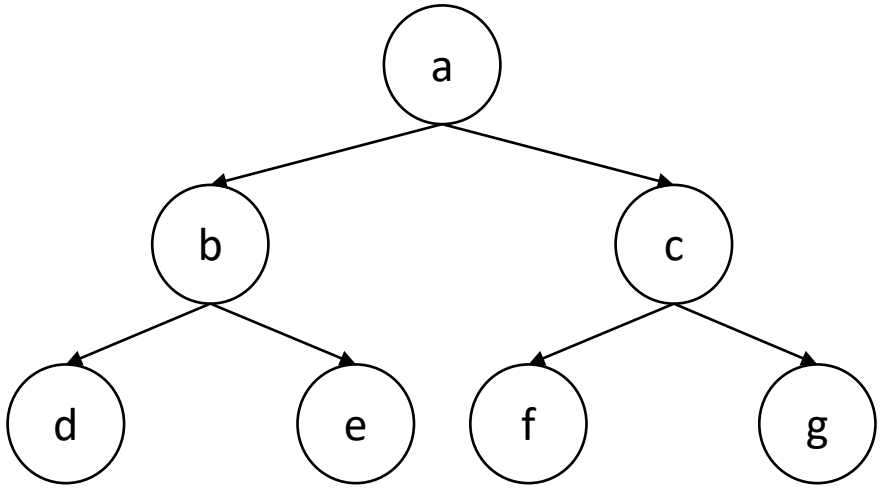
- Formally, we define a tree *T* as a set of nodes storing elements such that the nodes have a **parent-child** relationship that satisfies the following properties:
  - It *T* is non-empty, it has a special node, called the **root** of *T*, that has no parent.
  - Each node *v* of *T* different from the root has a unique parent node *w*; every node with parent *w* is a child of *w*.



Note that just like linked lists, a tree data structure may be empty (i.e., it contains no nodes).

Relationships Among Nodes

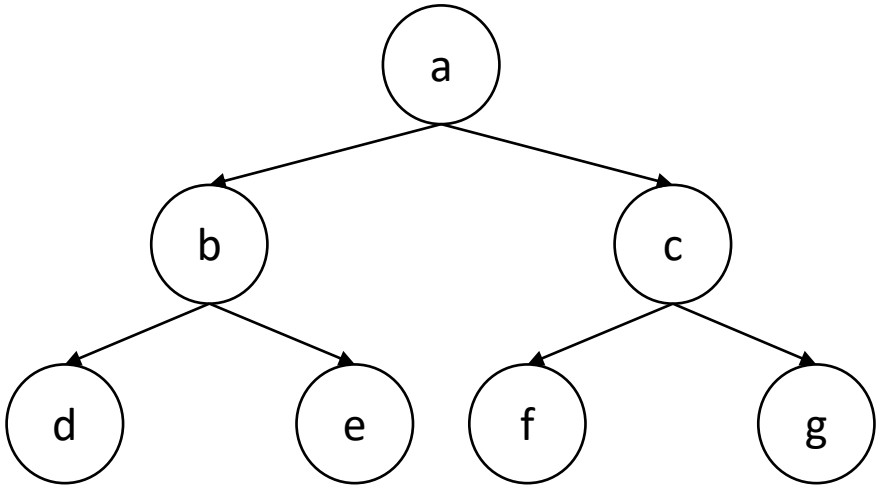
- **Siblings**: Nodes that are the children of the same parent.
- **Ancestor**: A node *u* is an ancestor of a node *v* if *u* = *v* or *u* is an ancestor of the parent of *v*.
- **Descendent**: A node *v* is a descendant of a node *u* if *u* is an ancestor of *v*.
- **Cousins**: Nodes whose parents are siblings.
- **Internal node**: A node with one or more children.
- **External node** (**leaf** node): A node with no children.



Siblings: b, c; d, e; f, g
Ancestors (of node e): e, b, a
Descendent (of node c): c, f, g
Cousins: d, f; d, g
Internal nodes: a, b, c
External/leaf nodes: d, e, f, g

Edges, Paths & Subtrees

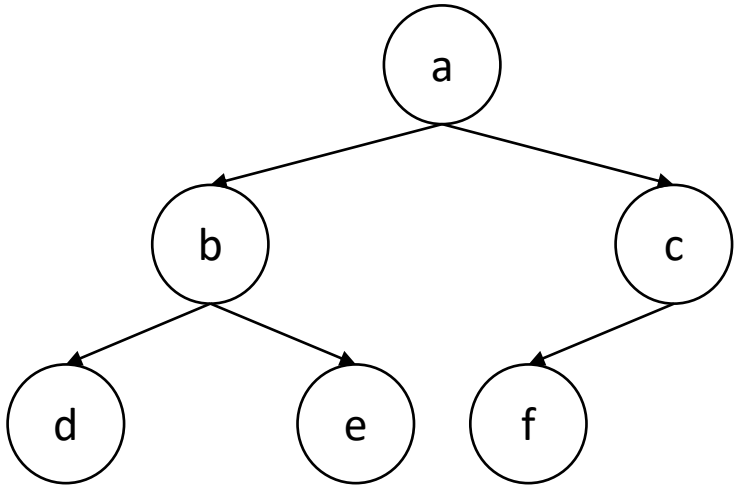
- **Edge**: An edge of tree *T* is a pair of nodes (*u*, *v*) such that *u* is the parent of *v*, or vice versa.
- **Path**: A path of tree *T* is a sequence of nodes such that any two consecutive nodes in the sequence form an edge.
- **Subtree**: The subtree of *T* rooted at a node *v* is the tree consisting of all the descendants of *v* in *T* (including *v* itself).



Edge: (a,b), (a,c)
Path: (a,c,f), (a,b,e)
Subtree: the subtree rooted a node c comprises of the tree with root c and children f and g

Properties of trees

- **Depth**: The depth of a node u is defined as:
  - 0 if *u* is the root node
  - 1 + the depth of the parent of *u*
- **Height**: The height of a node u is defined as:
  - 0 if *u* is a leaf node
  - 1 + maximum of the heights of *p*'s children
- **Height** (tree): The height of a (non-empty) tree is the height of the root node.
- **Level**: The level of a node is the same as its depth.
- **Degree**: The degree of a node is the number of its children.
- **Ordered tree**: A tree is ordered if there is a meaningful linear order among the children of each node.



Depth/level: a – 0; b, c – 1; d, e, f – 2
Height: d, e, f – 0; b, c – 1; a – 2
Degree: a, b – 2; c – 1, d, e, f - 0

Binary Tree

- A **binary tree** is an **ordered** tree with the following properties:
  - Every node has at most two children.
  - Each child node is labeled as being either a left child or a right child.
  - A left child precedes a right child in the order of children of a node.

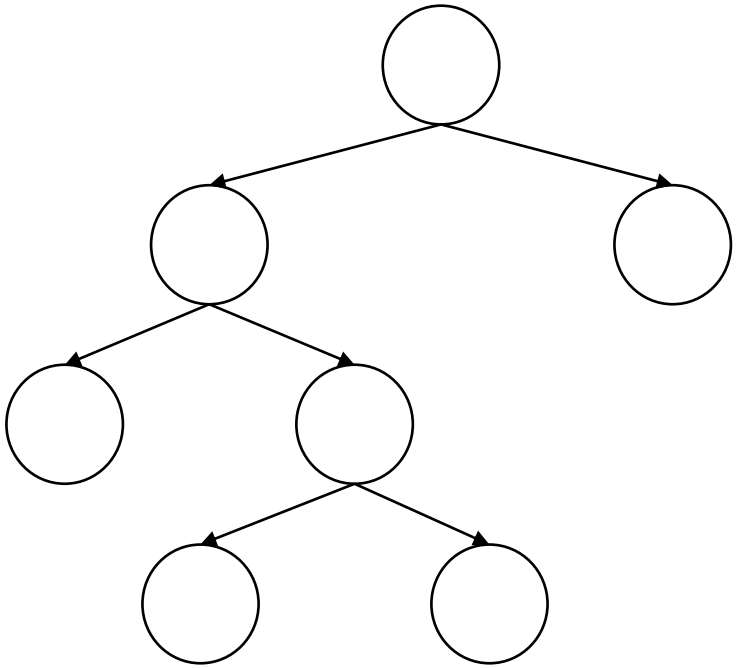
Types of Binary Trees

- Proper (full, strict)
- Perfect
- Complete
- Degenerate

- Note that some authors refer to Perfect trees as Complete trees, in which case, they refer to Complete trees as Almost Complete or Nearly Complete trees. It should be clear from the context, which terminology is being used in any scenario.

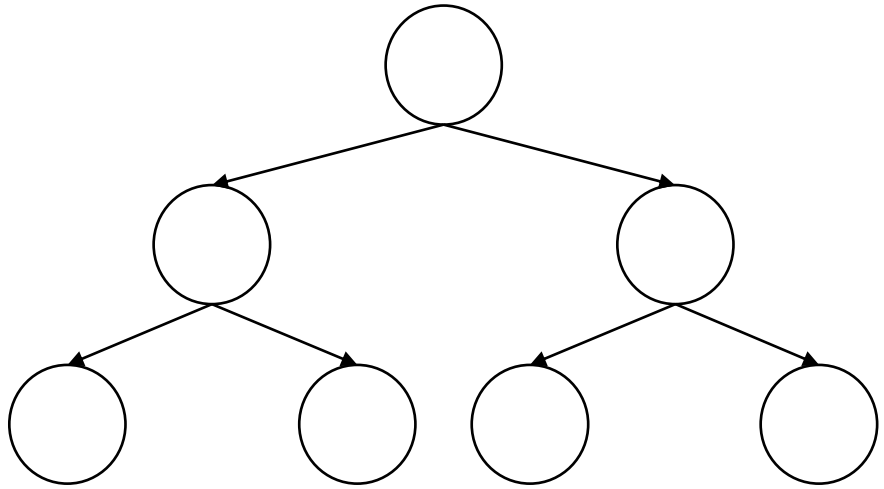
Proper Binary Tree

- Proper (full, strict): A binary tree in which each node has either 0 or 2 children.



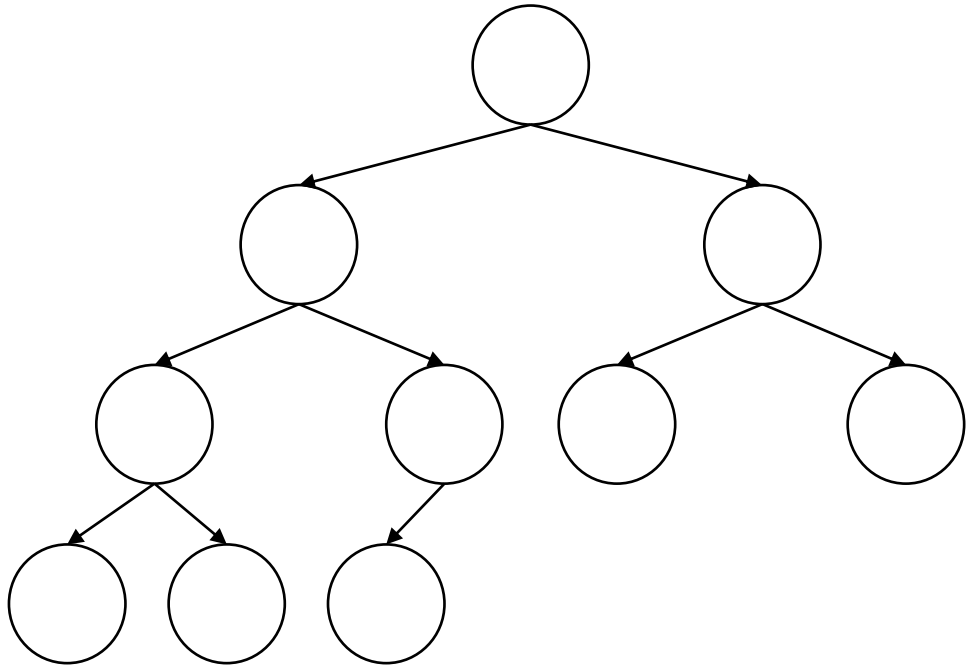
Perfect Binary Tree

- Perfect: A binary tree in which all interior nodes have 2 children and all leaves have the same depth (or level).



Complete Binary Tree

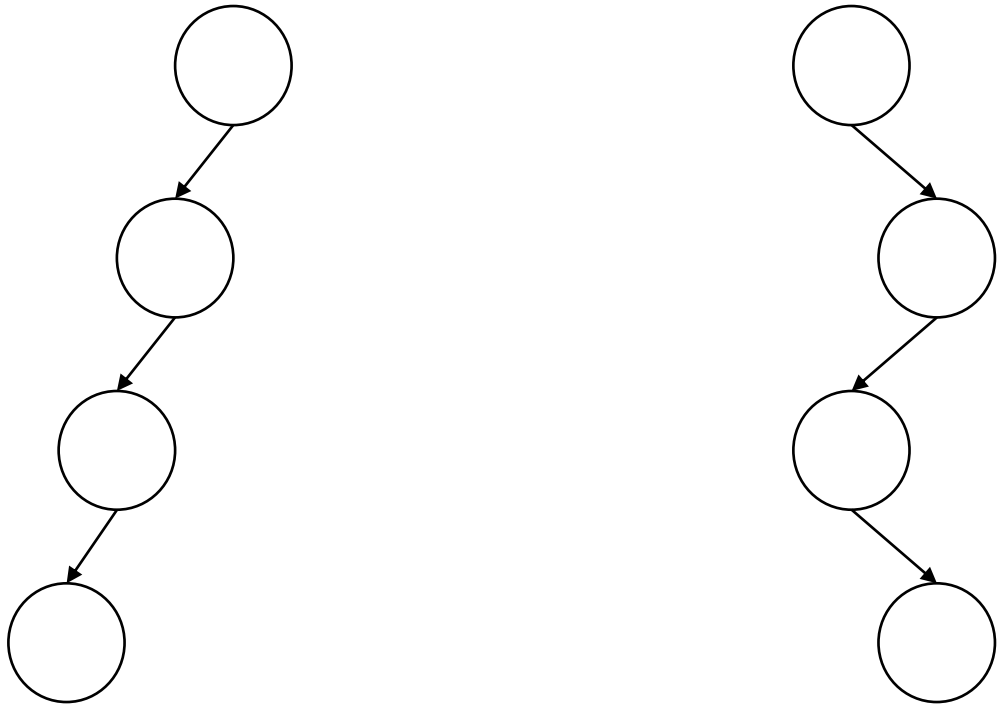
- Complete: A binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.



- A perfect tree is always complete, but a complete tree is not necessarily perfect.

Degenerate Binary Tree

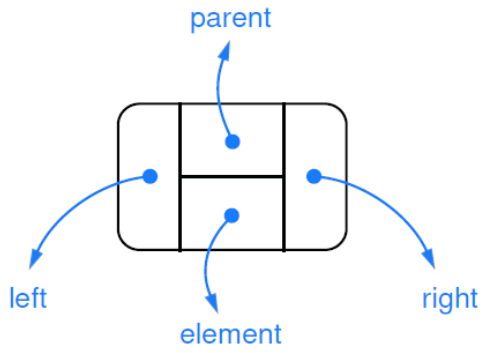
- Degenerate: A binary tree in which each parent node has only one associated child node. (Behaves like a linked list).



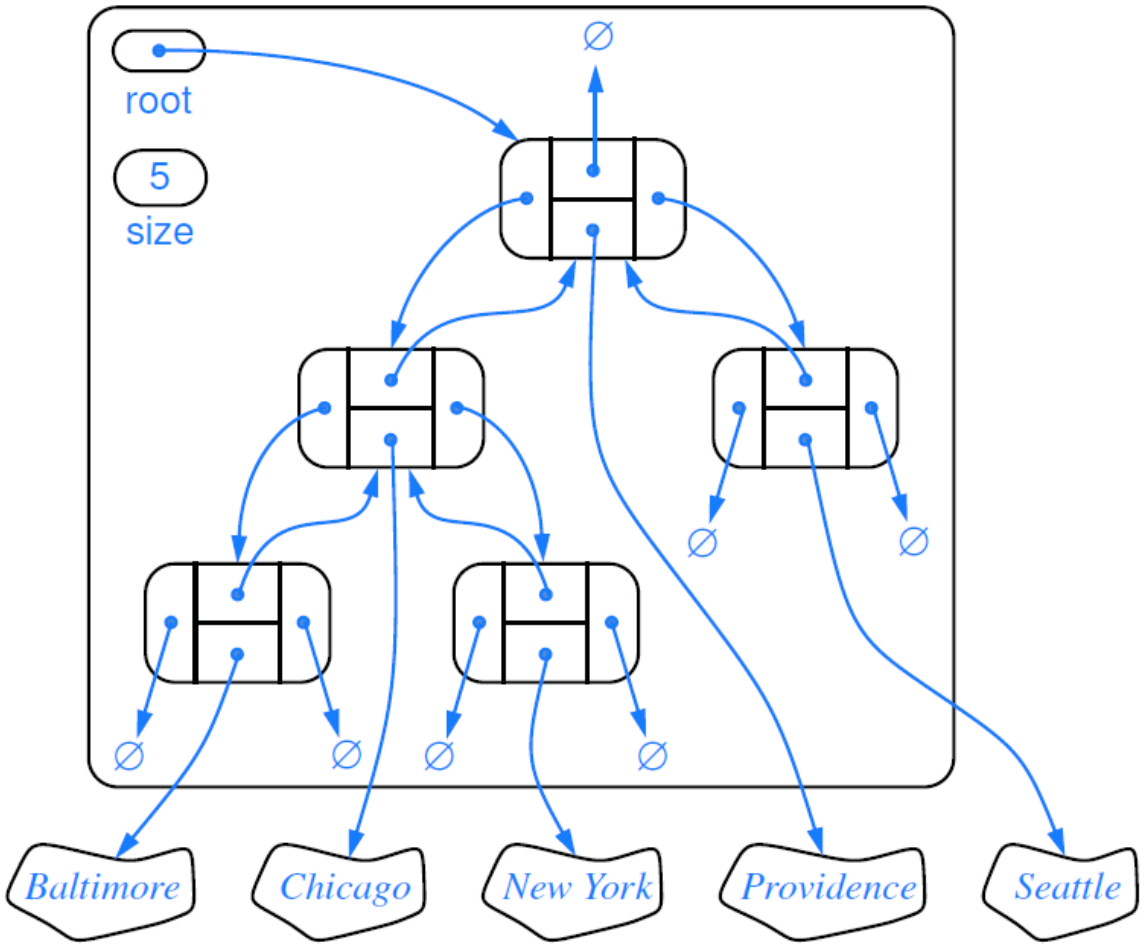
Implementing Binary Trees

- Like with stacks and queues, binary trees can also be implemented in two ways.
  - Using a **linked** structure (similar to singly-linked lists)
  - Using **arrays**.

Linked Structure for Binary Trees

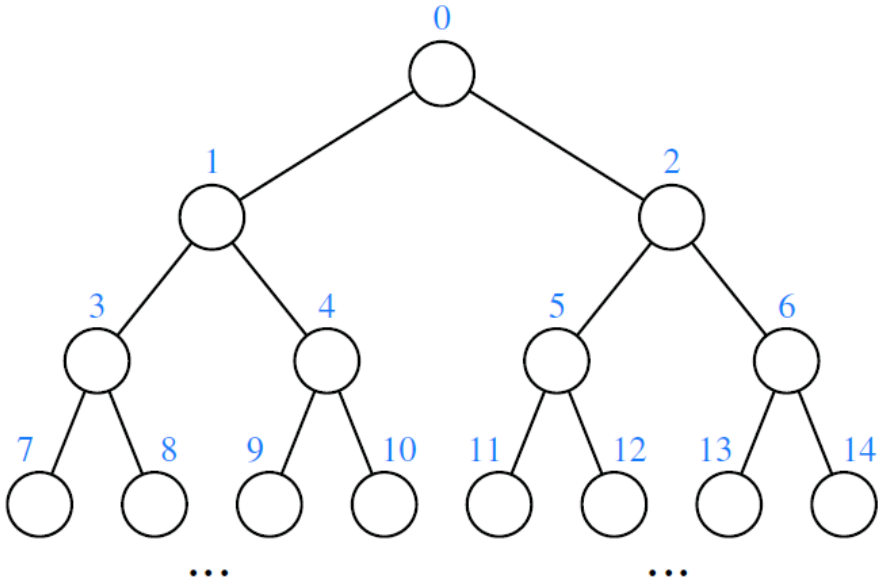



```
class TreeNode<T> {
        T element;
        TreeNode parent;
        TreeNode left;
        TreeNode right;
}
```

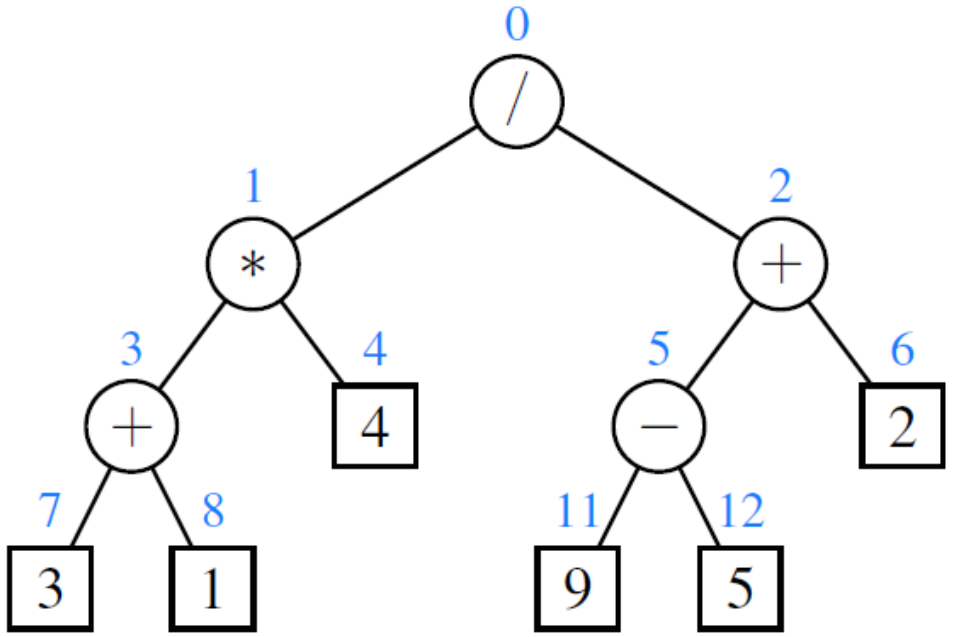


Array-based Representation for Binary Trees

- For every node p of tree T, let $f(p)$ be defined as follows:
  - If p is the root of T, then $f(p) = 0$.
  - If p is the left child of node q, then $f(p) = 2f(q) + 1$.
  - If p is the right child of node q, then $f(p) = 2f(q) + 2$.
- The numbering function f is known as the *level numbering* function. This function can be used to generate the array index for each node.



Array-based Representation for Binary Trees



/	*	+	+	4	−	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Tree Traversal Techniques

- A *traversal* of a tree *T* is a systematic way of accessing, or “visiting” all the nodes of *T*.
- The specific action to be performed on each node depends upon the application. For example, we might be incrementing the salaries of all employees stored in a binary tree.
- Tree traversal schemes:
  - Pre-order
  - Post-order
  - In-order

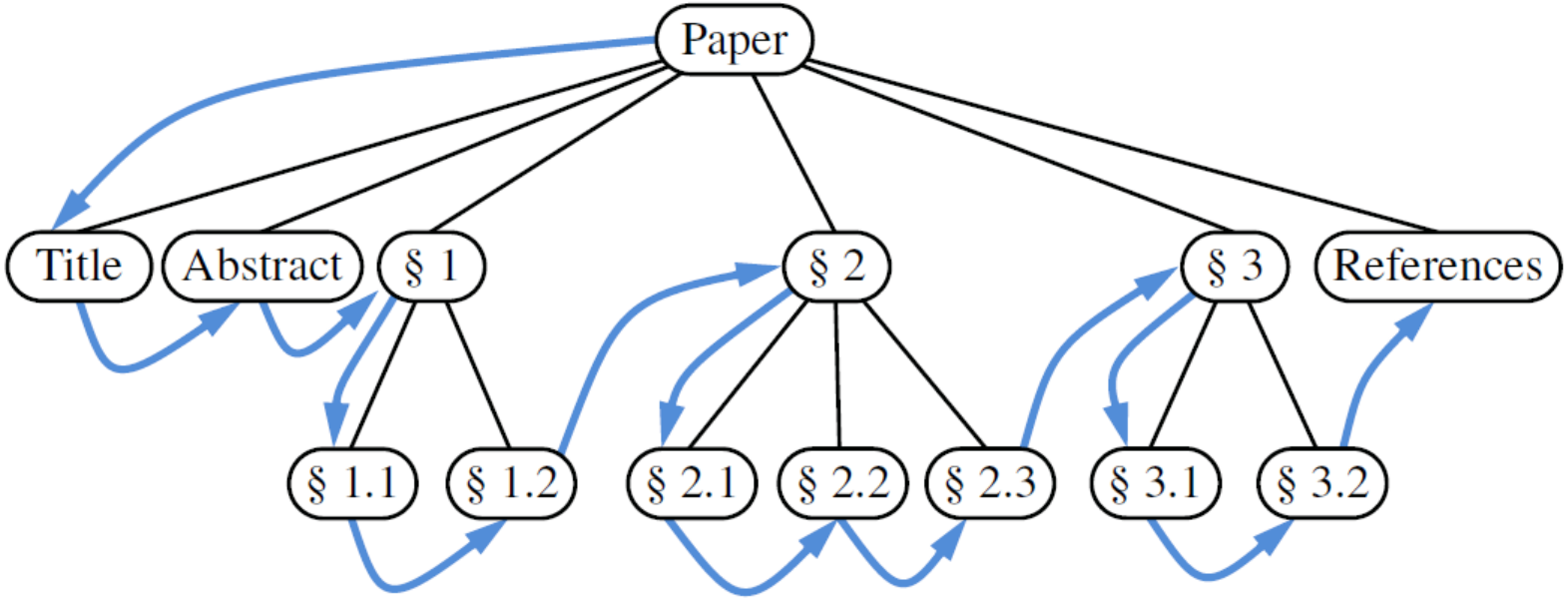
Pre-order Traversal

Algorithm pre-order(*p*):

perform “visit” action for node *p*

for each child *c* in the ordered list children(*p*) **do**

pre-order(*c*)



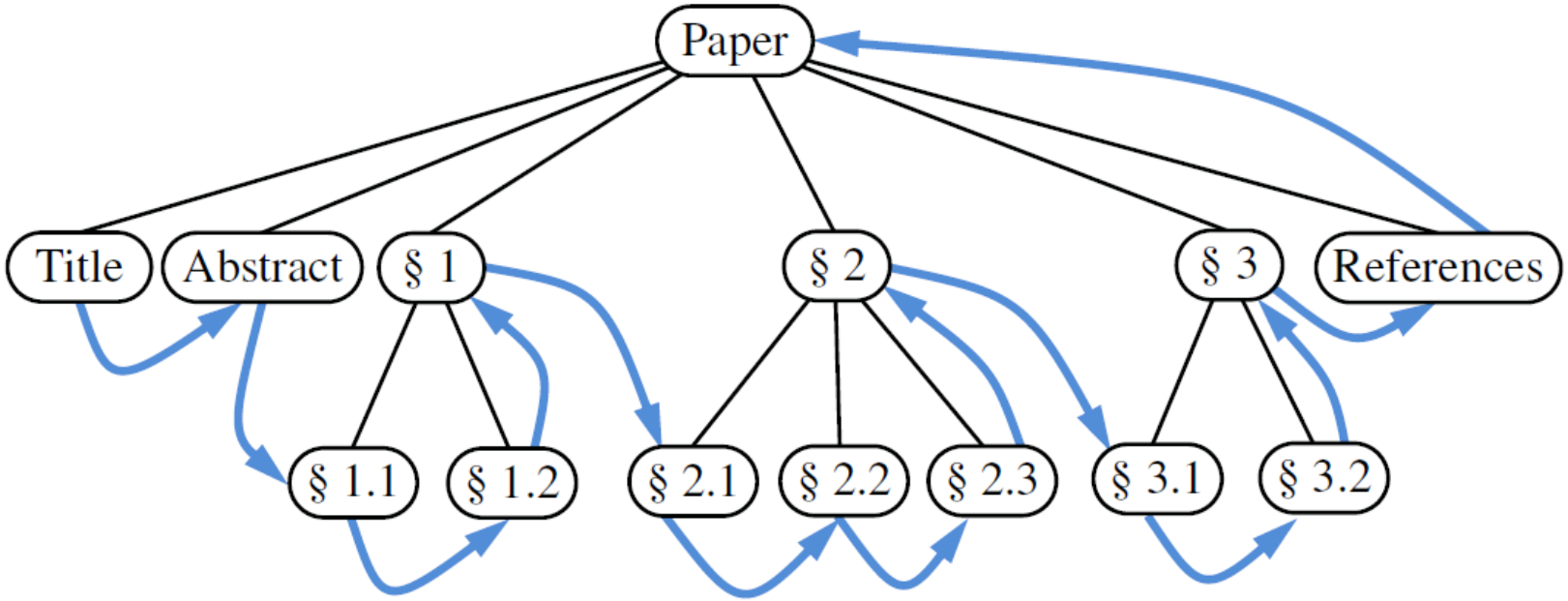
Post-order traversal

Algorithm post-order(*p*):

for each child *c* in the ordered list children(*p*) **do**

pre-order(*c*)

perform “visit” action for node *p*



In-order Traversal (Binary Tree)

Algorithm in-order(*p*):

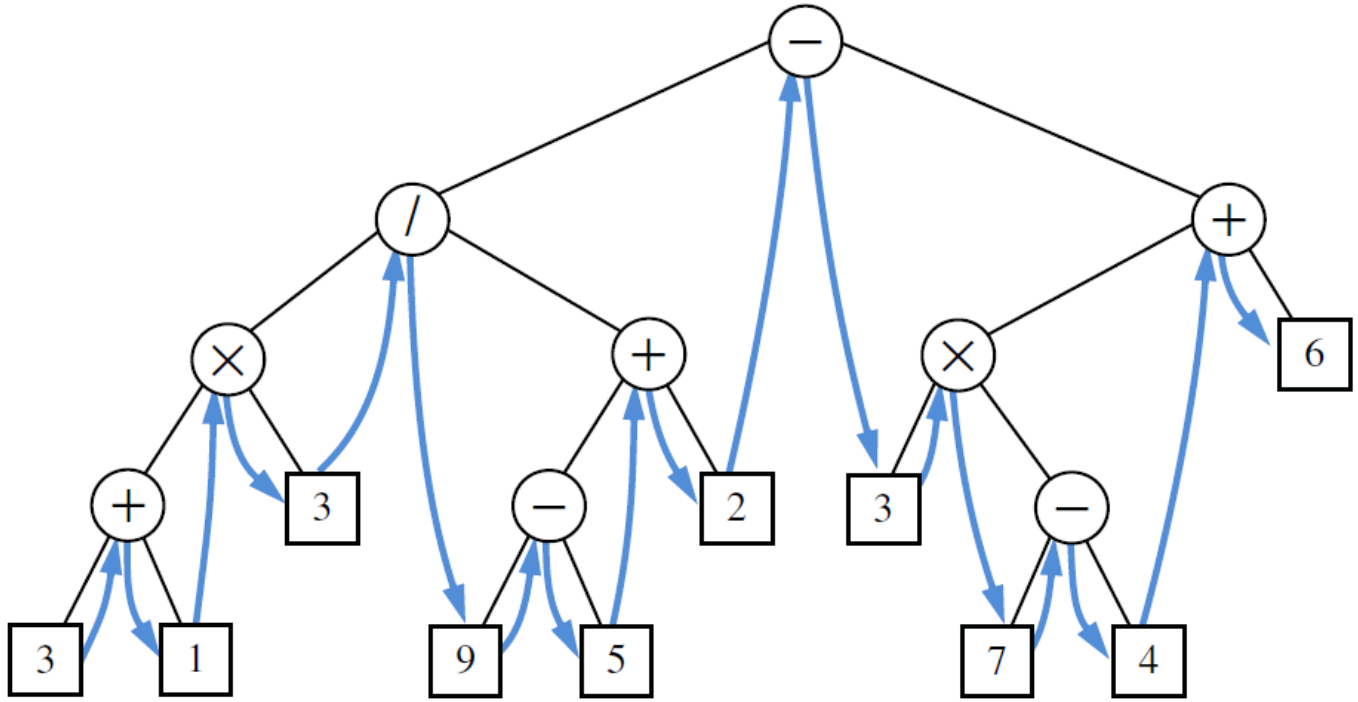
if *p* has a left child *lc* **then**

in-order(*lc*)

perform “visit” action for node *p*

if *p* has a right child *lr* **then**

in-order(*lr*)



Breadth First Traversal (or BFS Traversal)

Algorithm breadthfirst():

Initialise queue *Q* to contain root node

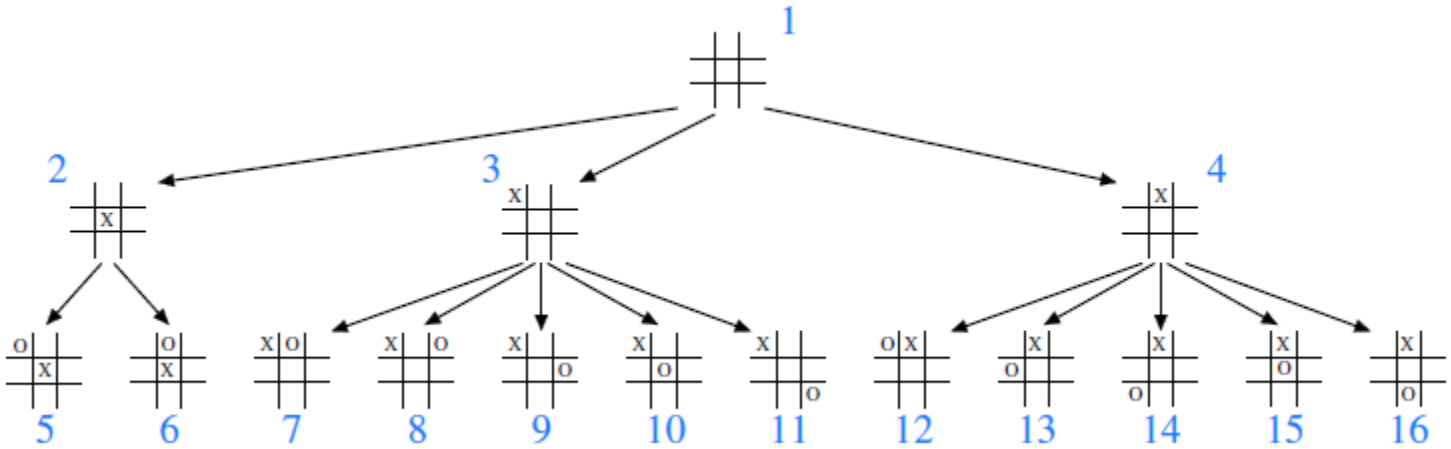
while *Q* not empty **do**

p = *Q*.dequeue()

perform “visit” action for node *p*

for each child *c* in the ordered list children(*p*) **do**

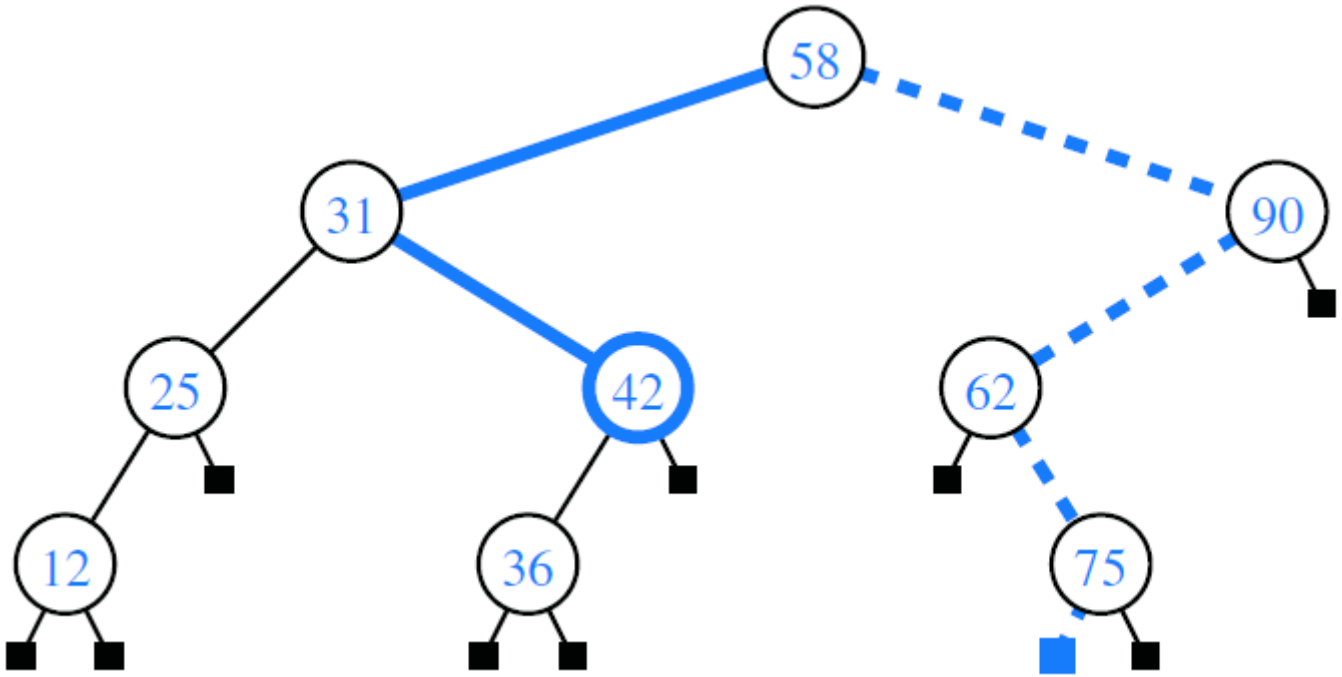
Q.enqueue(*c*)



In the tree shown above, each node consists of a tic-tac-toe game configuration. Ignoring the contents of the nodes, note that when using breadth first traversal, the order in which the nodes are visited correspond to a left-to-right order among increasing levels of the tree (the numbers in blue for each node corresponds to the order in which the nodes are visited using this traversal technique).

Binary Search Tree

- A Binary Search Tree (BST) is a binary tree where the element of each internal node is greater than the all the elements of the respective node's left subtree and less than the ones in its right subtree.



Properties of Binary Trees

Q. What is the maximum number of nodes in a binary tree of height h?

Sol.

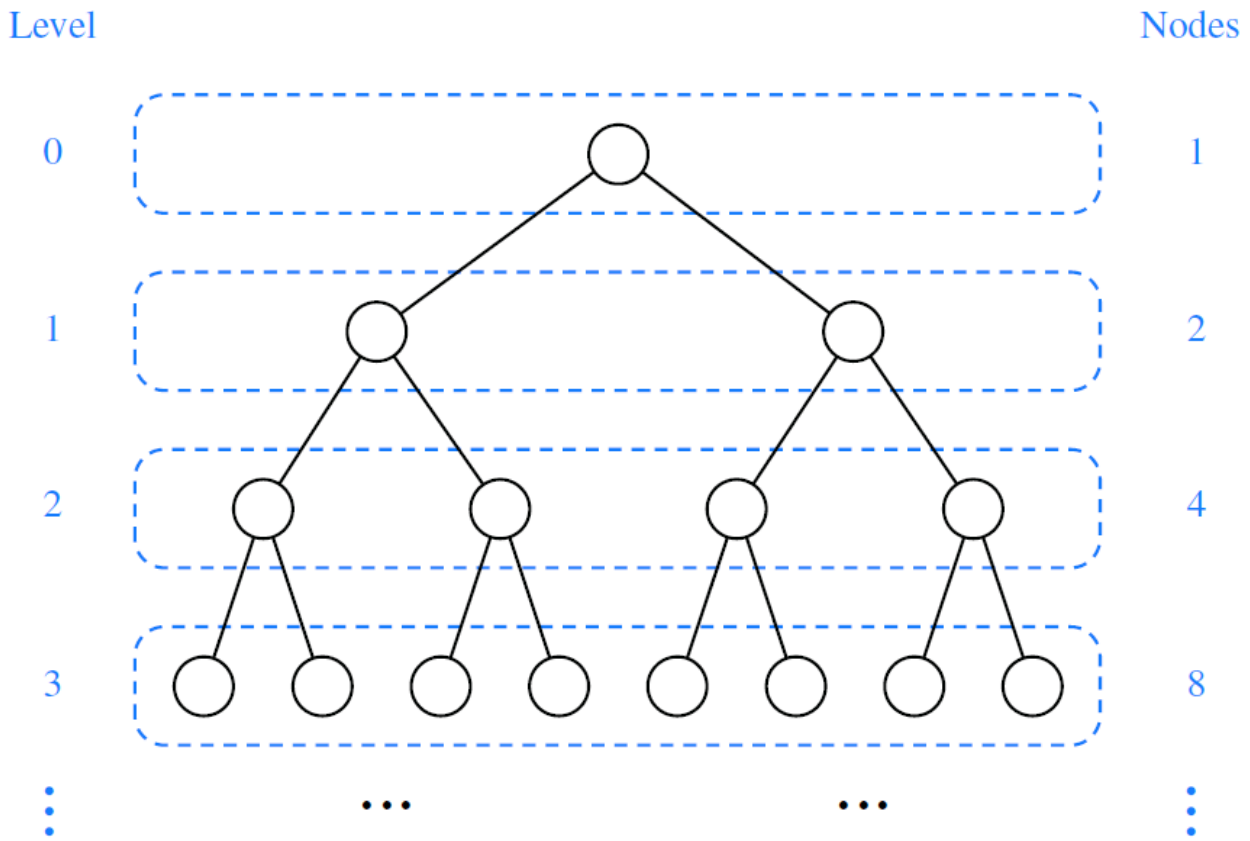
Maximum number of nodes

=> perfect binary tree

No. of nodes = $2^0 + 2^1 + 2^2 + \ldots + 2^h$

$= 1*((1-2^{h+1})/(1-2))$

$= 2^{h+1} - 1$



Properties of Binary Trees

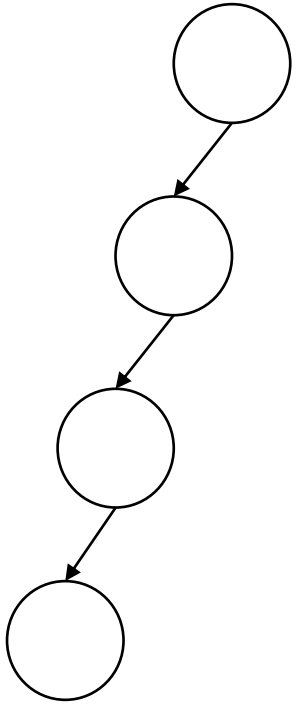
Q. What is the minimum number of nodes in a binary tree of height *h*?

Sol.

Minimum number of nodes

=> degenerate tree

No. of nodes = *h* + 1



Properties of Binary Trees

- Similarly, calculate the minimum and maximum nodes for
  - proper binary tree ($2h+1$, $2^{h+1}-1$)
  - perfect binary tree (2^{h+1}, $2^{h+1}-1$)
  - complete binary tree (2^{h}, $2^{h+1}-1$)
- Additional sample questions on properties of binary trees:
  - What is the number of leaf nodes in a perfect binary tree with *n* nodes?
  - What is the min and max possible height of a binary tree with *n* nodes?
  - What is the minimum height of a binary tree with *n* leaf nodes?
  - What is the relationship between the number of nodes and edges in a non-empty binary tree?
  - What is the minimum number of internal nodes in a complete binary tree of height *h*?